trutz.io

SPICKZETTEL · DOCKER ARCHITEKTUR

Ein Container ist nur ein Prozess

Namespaces, cgroups, die Kette hinter docker run und der containerd image store, auf einen Blick. Begleitmaterial zum Blog-Beitrag auf trutz.io.

DER MERKSATZ, AUS DEM ALLES FOLGT

Am Ende der Kette steht kein Hypervisor, sondern ein Prozess.

SÄULE 1

Tiefes technisches Verständnis

Wer weiß, dass ein Container ein Prozess ist, sucht Fehler dort, wo sie entstehen: im Kernel des Hosts, nicht in einer Blackbox.

SÄULE 2

Offene Standards

Images folgen dem OCI-Standard, containerd und runc sind eigenständige Open-Source-Projekte. Dein Wissen und deine Images gehören keinem Hersteller.

01 Die drei Kernel-Bausteine

Sichtschutz · Budget · Dateien

BAUSTEIN	WIRKUNG
Namespaces	was der Prozess **sieht**: eigene PIDs, Netzwerk, Mounts, Hostname
cgroups	was er **verbrauchen darf**: CPU, RAM, IO
Wurzeldateisystem	seine **Dateien**: der Inhalt des Docker Images

> ABGRENZUNG
>
> Kein Gast-Kernel, kein Hypervisor: alle Container teilen sich den Kernel des Hosts. Darum starten sie in Millisekunden und isolieren schwächer als VMs.

02 Die Kette hinter docker run

von oben nach unten

```
docker CLI    # nur ein Client, spricht REST
dockerd       # Koordinator: API, Netze, Volumes
containerd    # Images: Pull und Store
runc          # erzeugt den isolierten Prozess
```

Kubernetes steigt direkt bei containerd ein. Deshalb braucht ein Cluster kein Docker, und deine Images laufen trotzdem.

03 Der Beweis in drei Kommandos

auf jedem Rechner nachprüfbar

```
docker container run -d --name beweis nginx:latest
docker container top beweis
# außen: nginx mit normaler Host-PID
docker container exec beweis cat /proc/1/comm
# innen: PID 1 ist nginx, das ist der PID-Namespace
```

Aufräumen: `docker container rm -f beweis`

04

seit Engine 29 Standard

containerd image store: dran denken

- Standard bei Docker Desktop seit **4.34**, bei neuen Docker Engines seit **Version 29**; Upgrades bleiben auf `overlay2`, bis du umschaltest.
- Prüfen mit `docker info`: Storage Driver `overlay2` = klassisch, `driver-type: io.containerd.snapshotter.v1` = containerd image store.

```
# /etc/docker/daemon.json, dann Daemon neu starten
{ "features": { "containerd-snapshotter": true } }
```

GETRENNTE STORES

Beim Umschalten verschwinden bestehende Images nicht, sie liegen im jeweils anderen Backend und werden erst beim Zurückschalten wieder sichtbar.

Verstehen ist Säule eins. Weitergeben erlaubt.

Dieser Spickzettel gehört zum Blog-Beitrag über die Docker Architektur auf trutz.io. Jede Woche praxisnahe Antworten auf Docker- und Kubernetes-Probleme: der Newsletter.

trutz.io

Newsletter